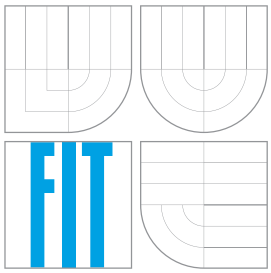
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ŘEŠENÍ ÚLOH S NEURČITOSTÍ

SOLVING PROBLEMS WITH UNCERTAINTY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LIBOR HRDÝ
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. FRANTIŠEK V. ZBOŘIL, CSc.
SUPERVISOR

BRNO 2007

Zadání bakalářské práce

Řešení úloh s neurčitostí

Solving Problems with Uncertainty

Vedoucí:
Zbořil František V., doc. Ing., CSc., UITS FIT VUT
Oponent:
Martinek David, Ing., UITS FIT VUT
Přihlášen:
Hrdý Libor
Zadání:
1. Prostudujte metody řešení her s neurčitostí.
2. Vyberte hru, resp. hry pro demonstraci.
3. Navrhněte programy.
4. Programy implementujte.
5. Proveďte experimenty.
6. Zhodnoťte dosažené výsledky

Kategorie:
Umělá inteligence
Implementační jazyk:
C, C++
Literatura:
- Podle pokynů vedoucího práce

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Abstrakt

V dokumentu je popsána implementace logické-deskové hry Vrhcáby (anglicky Backgammon), hry pro dva hráče, přičemž jeden z hráčů je zastoupen počítačem. V dokumentu je rozvedena problematika programování grafického uživatelského rozhraní pomocí toolkitu WxWidgets a dále implementace herního jádra (ovládání hry + UI počítače) s použitím algoritmu ExpectMiniMax, jež se využívá právě pro implementaci her jako Vrhcáby, tedy her, v nichž se vyskytuje prvek náhody, v tomto konkrétním případě hod kostkou.

Klíčová slova

WxWidgets, GUI, UI, Alfa-beta, MiniMax, ExpectMiniMax, náhoda, hry s neurčitostí, neurčitost, C++

Abstract

In this thesis is described implementation of the logical deskgame Backgammon, which is a game for two players, whereas one is substituted by computer. This thesis is focused on the problems of the programming the graphical user interface with help of toolkit WxWidgets and also the implemetnation of the game core (game controls and AI of the computer) by using ExpectMiniMax algorithm, that is used for the implementation of the games with the strong influence of random, games where random plays a big role, in this particular case throwing the cube.

Keywords

WxWidgets, GUI, UI, Alpha-beta, MiniMax, ExpectMiniMax, chance, games with uncerntainty, uncerntainty, C++

Citace

Libor Hrdý: Řešení úloh s neurčitostí, bakalářská práce, Brno, FIT VUT v Brně, 2007

Řešení úloh s neurčitostí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Františka Vítězslava Zbořila CSc. a že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Libor Hrdý
15. května 2007

Poděkování

Zde bych rád poděkoval panu doc. Ing. Františku Vítězslavu Zbořilovi CSc. za pomoc a rady, které mi poskytl při řešení této práce a také za studijní oporu pro předmět *Základy umělé inteligence*, která se při práci na totmo projektu stala mým hlavním studijním materiálem.

Obsah

Kapitola 1

Úvod

Téma mé bakalářské práce je *Řešení úloh s neurčitostí* a spadá do kategorie umělé inteligence. V této dokumentaci, strukturované do dvou hlavních částí, v první z nich nazvané *Teoretický náhled* nejprve popisuji aktuální stav této práce. Dále se snažím ukázat jednotlivá odvětví vědního oboru UI. V úvodu se zaměřuji na oblast *Metody řešení úloh*, kterou specializuji v kapitole *Metody hraní her*, kde jsou popsány některé základní metody hraní her např. procedury *MiniMax* a *Alfa-beta prořezávání*. V samotném závěru této sekce se dostávám k popisu problematiky neurčitosti a k popisu algoritmu *ExpectMiniMax*, který se často používá právě ve spojení s problematikou neurčitosti.

Na začátku druhé části popisuji v kapitole *Výběr hry pro demonstraci* výběr demonstrační hry z dané množiny her obsahujících prvek náhody a také důvody, které mě vedly ke konečnému rozhodnutí vybrat si hru Vrhcáby (Backgammon). Pravidla zvolené hry jsou shrnuta v příloze A. Osvojení těchto pravidel bylo nutným krokem pro správnou analýzu a návrh vlastního řešení, které tvoří další dvě kapitoly praktické části. Za těmito dvěma body následuje v části s názvem *Grafické uživatelské rozhraní* popis GUI a jeho konečného návrhu. Součástí kapitoly je i realizace hrací plochy a objektů, které plocha obsahuje. V následující kapitole nazvané *Herní jádro* jsou popsány jednotlivé stavy hry, které se během hraní pravidelně střídají. Poslední součástí dokumentu je závěr, ve kterém se snažím zhodnotit odvedenou práci. Také uvádím možná rozšíření různých částí programu vedle přínosů, které mi řešeni této práce dalo.

Na konci dokumentu jsou vloženy dvě přílohy. Příloha A s pravidly hry Vrhcáby a příloha B obsahující adresaářovou strukturu na přiloženém CD, postup při překladu programu a stručný návod k jeho použití.

Kapitola 2

Charakteristika současného stavu

V rámci semestrálního projektu jsem prostudoval metody řešení her s neurčitostí a z nich jsem si vybral použití algoritmu *ExpectMiniMax* (viz 3.3.4). Pro demonstraci jsem si zvolil hru Vrhcáby (viz kapitola 4.1) a tuto jsem implementoval za použití jazyka C++.

Aktuálním stavem, který je popsán v této dokumentaci, je hratelná verze této hry. Jsou implementována všechna základní pravidla nutná ke hraní jedné hry, tzn. každá hra končí vítězstvím jednoho z hráčů, ale tyto výsledky nejsou dále nijak zpracovávány. Série her se závěrečným vyhodnocením je možné rozšíření, které bych rád doimplementoval již jako mimoškolní projekt. V této souvislosti bych též implementoval modul zvyšování hodnoty vítězství hry. K tomuto zvyšování se používá speciální sázecí kostka, která se nachází již v současné verzi, ale zatím není aktivní. Hra je ovládána výhradně myší.

Aktuální grafické uživatelské rozhraní (dále jen GUI) je vytvořeno pomocí toolkitu WxWidgets. GUI je tvořeno herním menu, stavovým řádek a interaktivní hrací plochou, která je dále rozdělena do menších oblastí (viz 4.4) reagujících na vstup uživatele. Každý objekt ve hře je reprezentován obrázkem ve formátu png. Tento přístup umožňuje jednoduše změnit vzhled hry. Dalším možným rozšířením je právě volba vzhledu hrací desky, hracích kamenů, či kostek ve hře.

Do GUI jsem následně zakomponoval herní jádro, které je implementováno pro hru hráče (uživatele) proti počítači. Skládá se z dílčích funkcí, které řeší jednotlivé problémy v průběhu hry. Jedná se např. o zjištění korektního tahu v dané situaci nebo nalezení nejlepšího tahu pro hráče ovládaného počítačem. Nalezení nejlepšího tahu tvoří vlastní *umělou inteligenci* počítače a je implementováno pomocí algoritmu *ExpectMiniMax* (viz 3.3.4). I tuto oblast je možné rozšířit a sice aplikací procedury *alfa-beta prořezávání* (viz 3.5) na algoritmus *ExpectMiniMax*.

Uvažovaným rozšířením je např. i zvýraznění kamene, jímž se právě táhne, nebo kostky, jejíž hodnota je právě k tahu používána. Dále pak implementace modulu umožňujícího krok zpět.

Kapitola 3

Teoretický náhled

V této části dokumentace se nejprve zmíním o *umělé inteligenci* (dále jen UI) coby vědním oboru a o jejím současném rozdělení do dílčích odvětví. Následuje náhled na oblast *metody řešení úloh*, z nichž se zaměřím na *metody hraní her*.

3.1 Umělá inteligence

Na UI se můžeme dívat buď jako na vlastnost uměle vytvořeného systému (viz 3.1.1) nebo jako na vědní disciplínu (viz 3.1.2).

Definice 3.1.1 *"Umělá inteligence je vlastnost člověkem uměle vytvořených systémů vyznačujících se schopností rozpoznávat předměty, jevy a situace, analyzovat vztahy mezi nimi a tak vytvářet vnitřní modely světa, ve kterých tyto systémy existují, a na tomto základě pak přijímat účelná rozhodnutí, předvídat důsledky těchto rozhodnutí a objevovat nové zákonitosti mezi různými modely nebo jejich skupinami."* [3]

Definice 3.1.2 *"Umělá inteligence je věda o vytváření strojů nebo systémů, které budou při řešení určitého úkolu užívat takového postupu, který - kdyby ho dělal člověk - bychom považovali za projev jeho inteligence."* [4]

Dále v textu bude UI chápána dle znění definice jako vědní obor, který nemá pevně vymezený předmět zkoumání ani teoretický základ. Jedná se spíše o soubor metod, teoretických přístupů a algoritmů, sloužících k řešení velmi složitých úloh. Výsledky těchto dílčích řešení slouží buď jiným vědním disciplínám k aplikaci nebo jako základ k formování nových vědních disciplín.

Současná UI se převážně zaměřuje na práci s nejistými a neúplnými informacemi, na tzv. *Softcomputig* (zahrnuje neuronové sítě, genetické algoritmy, fuzzy množiny a fuzzy logiku, hrubé množiny, chaos). Také je intenzívně zkoumána problematika distribuované UI, tj. problematika agentů a multiagentních systémů. Tyto úlohy, kterými se UI zabývá, mohou do jisté míry nahradit některé intelektuální činnosti člověka a tím se UI rychle rozšiřuje z výzkumných laboratoří do reálného světa.

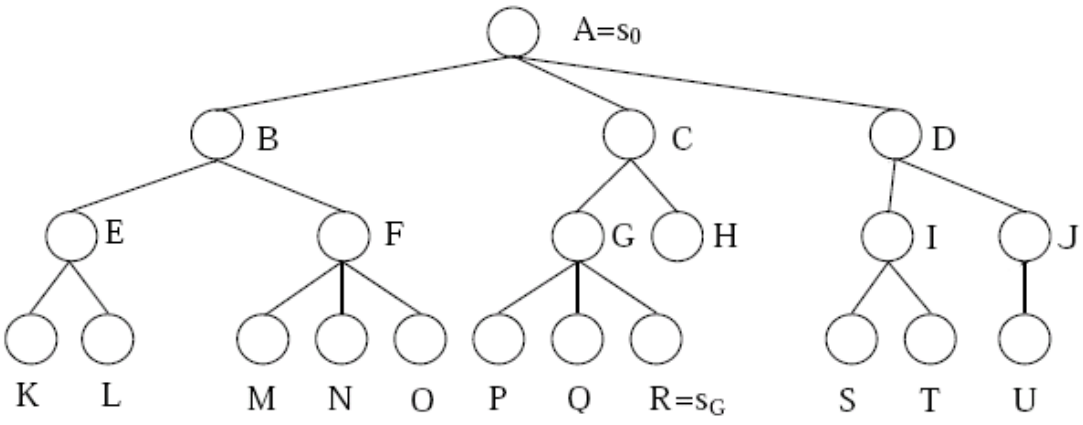
Aplikační oblasti UI v dnešní době jsou například tyto:

- Autonomní plánování (roboti na mimozemských objektech)
- Hraní her (šachy)
- Medicínská diagnostika (expertní systémy)
- Logické plánování (použito např. při plánování logistických operací během války v Perském zálivu v roce 1991)
- Porozumění přirozenému jazyku (spotřebiče ovládané hlasem)
- Neuronové sítě (snaha o napodobení lidského mozku)
- Automatické řešení složitých úloh (např. luštění křížovek)

3.2 Metody řešení úloh

Metody řešení úloh jsou významnou problematikou představující jednu z hlavních oblastí klasické UI. Tato oblast se zabývá konstrukcí inteligentních systémů, jejichž důležitým rysem je schopnost vytvářet vnitřní model světa a pracovat s ním. Je-li dán počáteční a cílový model prostředí, je úkolem systémů UI vyhledat takovou posloupnost akcí, jejichž aplikací lze dojít od stavu počátečního do stavu cílového. Každému modelu odpovídá určitý stav prostředí, jejich množina spolu s množinou akcí (operátorů), které umožňují stavy úlohy měnit, tvoří **stavový prostor**. Cílových stavů může být více a mohou být popsány podmínkami, které musí splňovat.

Reprezentaci řešené úlohy (stavového prostoru) lze znázornit různým způsobem. Nejčastěji se používá znázornění orientovaným stromem s uzly, které představují jednotlivé stavy a hranami, které znázorňují přechody mezi těmito uzly. Na obrázku 3.1 je příklad stavového prostoru, kde uzel A je uzlem kořenovým, zároveň označuje i počáteční stav a je v hloubce 0. Uzly H, K až U jsou uzly listové a uzel R reprezentuje koncový stav v hloubce 3.



Obrázek 3.1: Ukázka stavového prostoru

Prvním úkolem při řešení každé úlohy je jednoznačná formulace jejích cílů a jednoznačná definice operátorů, která zahrnuje i případné podmínky omezující jejich použití. Metody řešení úloh nám pak nabízejí postupy, kterými lze cílové stavy, resp. posloupnosti operátorů vedoucí k cílovým stavům, nalézat a představují tak prostředky nepostradatelné ve všech aplikačních oblastech UI.

Při automatickém řešení úloh je třeba pro nalezení řešení použít vhodnou metodu (strategii). Pro každý typ problému se hodí jiné metody. Podle tohoto kritéria se dají metody řešení úloh rozdělit např. následovně:

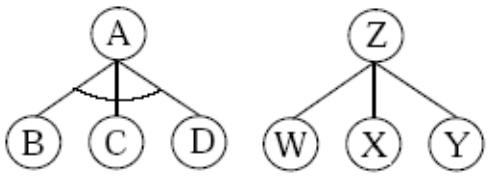
- Metody založené na prohledávání stavového prostoru (prohledávání do šířky, prohledávání do hloubky, backtracking aj.)
- Metody řešení úloh s omezujícími podmínkami (forward checking, metoda minimálního konfliktu)
- Metody založené na rozkladu úloh/problémů na podproblémy (použití AND/OR grafů)
- Metody hraní her (algoritmus MiniMax)

3.3 Metody hraní her

Metody hraní her uvedené v této kapitole jsou omezeny pouze na hry se dvěma (proti)hráči, kteří se po jednotlivých tazích hry pravidelně střídají. Oba hráči mají všechny informace o hře a jejím současném stavu. Tato informace je konečná (pozice) a obsahuje též údaj o tom, který hráč právě táhne. Dále existují pravidla hry, jenž určují ke každé pozici pro právě táhnoucího hráče konečný počet přípustných tahů. Krokem hry (tahem, popř. půltahem) je, když si táhnoucí hráč vybere jeden z přípustných tahů a provede jej. Tím vznikne nová pozice a na tahu je soupeř. Hra pokračuje, dokud se nedostane do závěrečné fáze, u které musí být též definováno, kdo vyhrál či prohrál, příp. že jde o remízu.

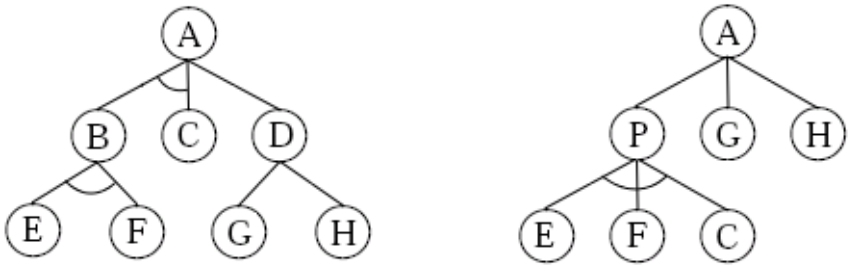
Dále zmíněné metody jsou založeny na rozkladu úlohy na podproblémy, což je přirozená metoda, kterou při řešení obtížných problémů používá i člověk.

Při rozkladu můžeme narazit na dva typy podproblémů (viz obrázek 3.2). Problém AND (na obrázku uzel A), který je řešitelný pokud jsou řešitelné všechny jeho podproblémy a problém OR (uzel Z), který je řešitelný, je-li řešitelný alespoň jeden z jeho podproblémů.



Obrázek 3.2: Podproblémy typu AND a OR.

Na obrázku 3.3 je vidět případné řešení nesourodosti problému na stejné úrovni zanoření, jež spočívá v přidání pomocného uzlu (na obrázku uzel P) či odebrání přebytečného uzlu (uzel D). Touto úpravou vznikne tzv. "čistý" AND/OR graf, který lze při řešení herních úloh procházet podobně jako stavový prostor.



Obrázek 3.3: Příklad převodu na "čistý" AND/OR graf

Při hraní her, respektive při aplikaci metod pro hraní her, je řešeným problémem nalezení tahu hráče, který právě táhne (hráč A). Z pohledu tohoto hráče bude problém řešitelný, pokud k jeho výhře povede alespoň jeden z jeho možných následujících tahů (problém OR). Při dalším kroku táhne protihráč (hráč B), který se snaží zabránit ve výhře hráči A. Držíme-li se pohledu hráče A, musí být všechny tahy hráče B následující po tahu hráče A pro hráče B neřešitelné, jinak řečeno, všechny tahy hráče B musí být řešitelné pro hráče A (problém AND). Hledání tahu vedoucího k výhře tak vede na klasické prohledávání AND/OR grafu.

Když se hráč A dostane znovu na řadu, nemůže použít výsledky z předchozího průchodu, protože na rozdíl od nového stavu hry nejsou tyto ovlivněny právě skončeným tahem hráče B, ale musí znovu najít a vybrat svůj tah již z nového konkrétního stavu úlohy.

Po vyhodnocení všech přípustných tahů si hráč, který táhne (uvažujme hráče A), vybere tah s nejlepším ohodnocením, které určuje tzv. "funkce zisku" nebo též "přínosnosti", kde toto ohodnocení odráží aktuální stav hry ve prospěch hráče A. Z toho plyne, že zatímco hráč A si vybírá tahy s největším ohodnocením, hráč B vybírá (respektive je toto při strojovém vyhodnocení uvažováno) nejméně ohodnocené tahy, aby hráči A co nejvíce znemožnil šanci na výhru.

U normálního vyhledávacího problému by hráč A prostě hledal sekvenci tahů vedoucích do vítězného konečného stavu (s použitím funkce přínosnosti) a po nalezení cesty by vybral první z tahů takové sekvence. Protihráč B ale hru ovlivňuje, proto musí hráč A najít strategii vedoucí ke konečnému stavu bez ohledu na to, co hráč B dělá.

Strategie obsahuje korektní tah (tah podle pravidel) pro hráče A jako odpověď na každý možný tah protihráče. Existují cesty k nalezení strategie, i když omezením pro výpočet může být (a bývá) časový limit.

Takto popsané hry lze rozdělit na jednoduché, složité a hry s neurčitostí.

3.3.1 Jednoduché hry

Za jednoduché hry se považují takové hry, u nichž je možné v reálném čase prohledat celý jejich AND/OR graf až do nalezení nějakého cílového stavu (výhra, prohra, remíza). K řešení se může použít například AO algoritmus, který je základním neinformovaným algoritmem. Podobně jako u prohledávání stavového prostoru můžeme i nyní vyšetřovat řešitelnost problému procházením AND/OR grafu do hloubky nebo do šířky (podle toho, který z uzlů vyjmeme ze seznamu OPEN).

AO algoritmus

1. Sestroj prázdné seznamy OPEN a CLOSED. Do seznamu OPEN ulož počáteční uzel (problém).
2. Vyjmi uzel zleva ze seznamu OPEN a označ jej jako uzel X.
3. (a) Pokud je uzel (problém) X řešitelný, přidej informaci o jeho řešitelnosti jeho předchůdcům. Je-li X zároveň počátečním problémem, ukonči řešení jako úspěšné - vytvoř a vrať relevantní část AND/OR grafu.
   (b) Není-li uzel (problém) X řešitelný a nelze-li jej rozložit na podproblémy, předej informaci o jeho neřešitelnosti jeho předchůdcům. Je-li X zároveň počátečním problémem, ukonči řešení jako neúspěšné.
   (c) Expanduj X (rozlož X na podproblémy) a všechny jeho následníky ulož do OPEN.
4. Ulož X do CLOSED.
5. Je-li seznam OPEN prázdný, ukonči řešení jako neúspěšné, jinak se vrať na bod 2.

S tím, že v případě řešitelnosti není nutné vracet celou část AND/OR grafu, ale pouze tah hráče A, který vede k jeho výhře.

3.3.2 Složité hry

Za složité hry považujeme hry, u kterých, pro velký počet uzlů v prohledávacím stromu (AND/OR grafu), není kompletní průchod tohoto stromu z časových důvodů reálný (např. šachy, Vrhcáby). K řešení takovýchto her se může použit např. algoritmus MiniMax.

Základem algoritmu MiniMax je rekurzivní procedura, označovaná také jako MiniMax, která se zavolá pro aktuální stav hry (kořenový uzel AND/OR grafu) a hráče, který je na tahu (hráč MAX). Tato procedura vrací ohodnocení uzlu a pro hráče MAX i tah k uzlu s maximálním ohodnocením, tj. tah, který je v daném stavu hry pro tohoto hráče nejvýhodnější. Samotné ohodnocení uzlu počítá již dříve zmiňovaná *funkce zisku*, která tak činí na základě převodu situace na hracím poli na jediné konkrétní číslo. Funkce zisku se pro každou hru liší a při jejím návrhu lze zohlednit např. počet figur, jejich "sílu", pozici na hrací ploše apod.

Na rozdíl od řešení jednoduchých her, kdy je možné projít prohledávacím stromem až k uzlu, který je koncovým stavem úlohy, zde procedura MiniMax předpokládá, že je zadána maximální hloubka prohledávání (počet zkoumaných tahů) a hodnoty terminálních uzlů tak neodráží výhru nebo porážku.[1]

Procedura MiniMax

1. Nazvěme předaný vstupní uzel uzlem X.
2. Je-li uzel X listem (konečným stavem hry nebo uzlem v maximální hloubce), vrať ohodnocení tohoto uzlu. Jinak pokračuj.
3. Je-li na tahu hráč MAX, tak postupně pro všechny jeho možné tahy (bezprostřední následníky uzlu X a hráče MIN) volej proceduru MiniMax a vrať maximální z navrácených hodnot. Je-li X kořenovým uzlem vrať i tah, který vede k nejlépe ohodnocenému bezprostřednímu následníkovi.
4. Je-li na tahu hráč MIN, tak postupně pro všechny jeho možné tahy (bezprostřední následníky uzlu X a hráče MAX) volej proceduru MiniMax a vrať minimální z navrácených hodnot.

Při použití procedury MiniMax dochází ke zbytečnému vyšetřování velké části AND/OR grafu (viz obrázek 3.4) z důvodu, který je blíže vysvětlen v následujícím příkladu na proceduru MiniMax. V obrázku jsou vyznačeny zbytečně vyšetřované uzly červeně, takových uzlů je přibližně 30% (13 ze 40ti). Tento příklad je z velké části převzat z opory předmětu IZU. [1]

Příklad použití algoritmu MiniMax

Hráč MAX (kořenový uzel A) volá proceduru MiniMax na svůj první tah a hráče MIN (uzel B) a ten volá tuto proceduru na svůj první tah a hráče MAX (uzel C). Hráč MAX (uzel C) volá proceduru MiniMax postupně na všechny své možné tahy a hráče MIN, protože všichni jeho bezprostřední následníci jsou uzlovými listy, procedura MiniMax pouze postupně vrací ohodnocení těchto listů a hráč MAX pak vybere (vrátí) maximální hodnotu z jejich ohodnocení (tj. číslo 8). Hráč MIN (uzel B) tuto hodnotu akceptuje a volá proceduru MiniMax na svůj druhý tah a hráče MAX (uzel D). První bezprostřední následník tohoto uzlu (listový uzel) vrací hodnotu 9.

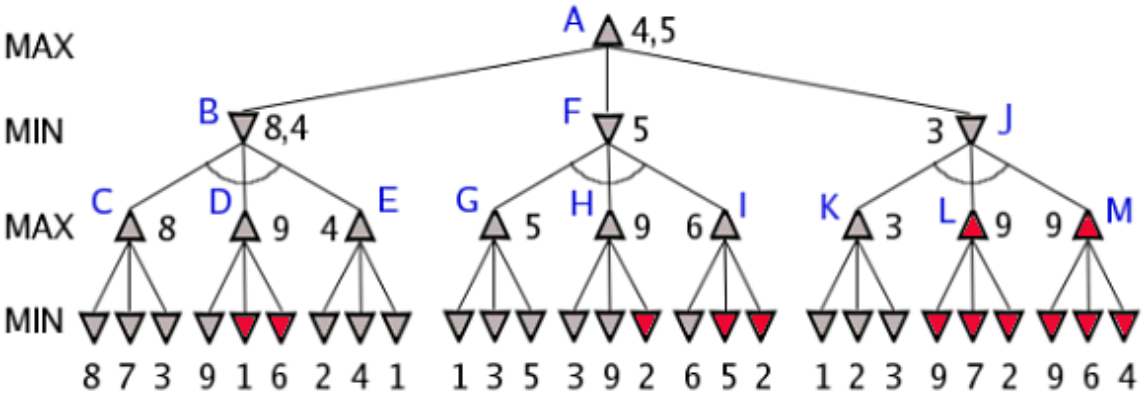
Je zřejmé, že prohledávání dalších následníků uzlu D je zbytečné, protože již nyní je jasné, že tento uzel vrátí hodnotu ≥ 9, a že hráč MIN (uzel B), který si vybírá tah s minimálním ohodnocením, si tento tah nevybere, protože ohodnocení jeho prvního tahu je menší.

Hráč MIN (uzel B) pak volá proceduru MiniMax na svůj poslední tah a hráče MAX (uzel E). Hráč MAX (uzel E) opět volá postupně proceduru MiniMax na všechny své bezprostřední následníky a z vrácených hodnot vybere (vrátí) hodnotu maximální, tj. hodnotu 4. Protože tato hodnota je menší než hodnota 8, hráč MIN (uzel B) si vybere a vrátí

[1] Při oceňování hry s omezenou hloubkou mohou někdy vést heuristicky slibné cesty později ve hře do špatné situace - jedná se o tzv. *horizont efekt*.

tuto hodnotu. Hráč MAX (uzel A) hodnotu 4 akceptuje a volá proceduru MiniMax na svůj druhý možný tah a hráče MIN (uzel F).

Další postup pro tento tah je velmi podobný postupu pro první tah. Hráč MIN (uzel F) volá postupně proceduru MiniMax na své bezprostřední následníky (uzly G, H a I) a vrátí minimum z vrácených hodnot, tj. číslo 5. Některé listové uzly, na obrázku označené červenou barvou, se opět vyšetřují zbytečně, z důvodů popsaných výše. Protože $5 > 4$, akceptuje hráč MAX (uzel A) tuto hodnotu (druhý tah je pro něj výhodnější, než tah první) a volá proceduru MiniMax na svůj třetí tah a hráče MIN (uzel J). Hráč MIN (uzel J) volá proceduru MiniMax na svůj první tah a hráče MAX (uzel K) a od tohoto uzlu dostane navrácenu hodnotu 3. Proto je již v tomto okamžiku zřejmé, že hráč MIN (uzel J), který si vybírá minimum, vrátí kořenovému uzlu hodnotu ≤ 3 a hráč MAX (uzel A) si tento tah nevybere. Další vyšetřování tahů hráče MIN (uzlu J) je tak zbytečné.



Obrázek 3.4: Ukázka použití algoritmu MiniMax pro jednoduché hry

3.3.3 Alfa-beta prořezávání

Podstata algoritmu *alfa-beta prořezávání* spočívá v nalezení "špatné větve" (tj. horší než doposud nalezená nejlepší) v prohledávacím stromu. Důležité přitom je, že nepotřebujeme vědět, jak moc špatná tato větev je. Navíc nás ani nezajímá, zda-li tam nebude lepší podvětev, protože soupeř by se jí jistě uměl vyhnout. Z porovnání algoritmů minimaxu a alfa-beta prořezávání vyplývá vlastnost, že algoritmus alfa-beta prořezávání vrací hodnotu, kterou by vrátil původní algoritmus, aniž by musel projít celým stromem, což je přesně to, co jsme potřebovali získat.

Při aplikaci algoritmu postupujeme tak, že si pamatujeme nějaké maximum (nebo minimum, pokud jde o minimalizující úroveň) pro množinu uzlů na jedné úrovni a pokud zjistíme, že ohodnocení synů dalšího uzlu v řadě je menší (větší) než toto maximum (minimum), nepotřebujeme tento uzel dál rozvíjet, protože víme, že už minimaxovou hodnotu svého otce neovlivní.

Procedura Alfa-beta

1. Je-li X počátečním/kořenovým uzlem, nastav $\alpha = -\infty$, $\beta = \infty$ (v praxi nastav hodnoty těchto proměnných na minimální a maximální možnou hodnotu).

2. Je-li uzel X listem (konečným stavem hry nebo uzlem v maximální hloubce) ukonči proceduru a vrať ohodnocení tohoto uzlu.

3. Je-li uzel typu AND (na tahu je hráč MIN) jdi na bod 4, jinak pokračuj (uzel je typu OR, na tahu je hráč MAX):

   (a) Dokud je $\alpha < \beta$, tak postupně pro první/další tah (bezprostředního následníka uzlu X a hráče MIN) volej proceduru Alfa-beta s aktuálními hodnotami proměnných α a β. Po každém vyšetřeném tahu nastav hodnotu proměnné a na maximum z aktuální a navrácené hodnoty .

   (b) Ukonči proceduru, vrať aktuální hodnotu proměnné α a pro kořenový uzel vrať i tah, který vede k nejlépe ohodnocenému bezprostřednímu následníkovi.

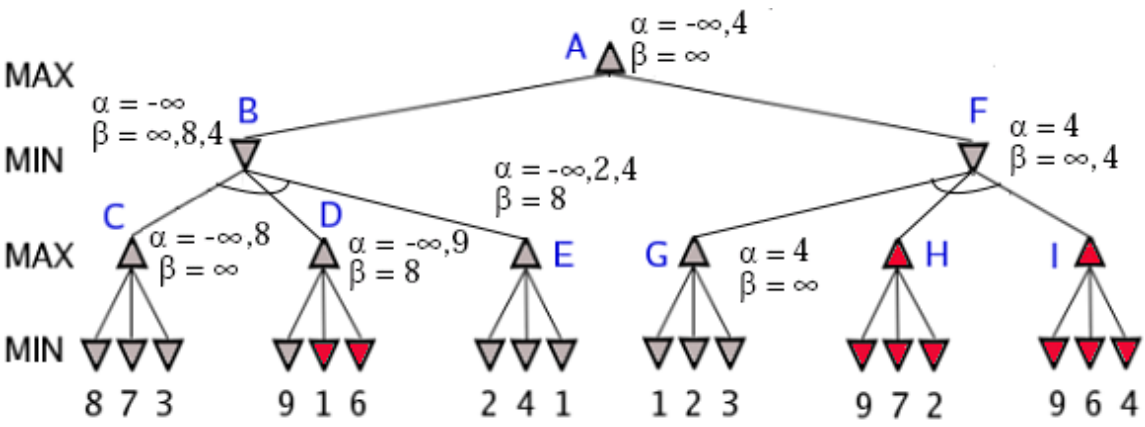
4. Uzel je typu AND (na tahu je hráč MIN):

   (a) Dokud je $\alpha < \beta$, tak postupně pro první/další tah (bezprostředního následníka uzlu X a hráče MAX) volej proceduru Alfa-beta s aktuálními hodnotami proměnných α a β. Po každém vyšetřeném tahu nastav hodnotu proměnné β na minimum z aktuální a navrácené hodnoty .

   (b) Ukonči proceduru a vrať aktuální hodnotu proměnné β.

Na obrázku 3.5 je vidět použití procedury Alfa-beta. Princip průchodu je tu stejný jako u algoritmu MiniMax 3.4 s tím, že se "prořežou" větve, které by se při použití algoritmu MiniMax procházeli zbytečně.

Podíváme-li se v obrázku na uzel C, hráč MAX si zde vybere ze svých synovských uzlů ten s maximálním ohodnocením (8). Tuto hodnotu vrátí rodičovskému uzlu (B) a hráči MIN, který ji akceptuje a zavolá proceduru Alfa-beta pro svůj další synovský uzel D a hráče MAX. V uzlu D se hráč snaží opět vybrat maximum ze svých synovských uzlů, ale již u prvního narazí na hodnotu, která je vyšší než hodnota, kterou vrátil z uzlu C. Z pohledu



Obrázek 3.5: Ukázka použití procedury Alfa-beta

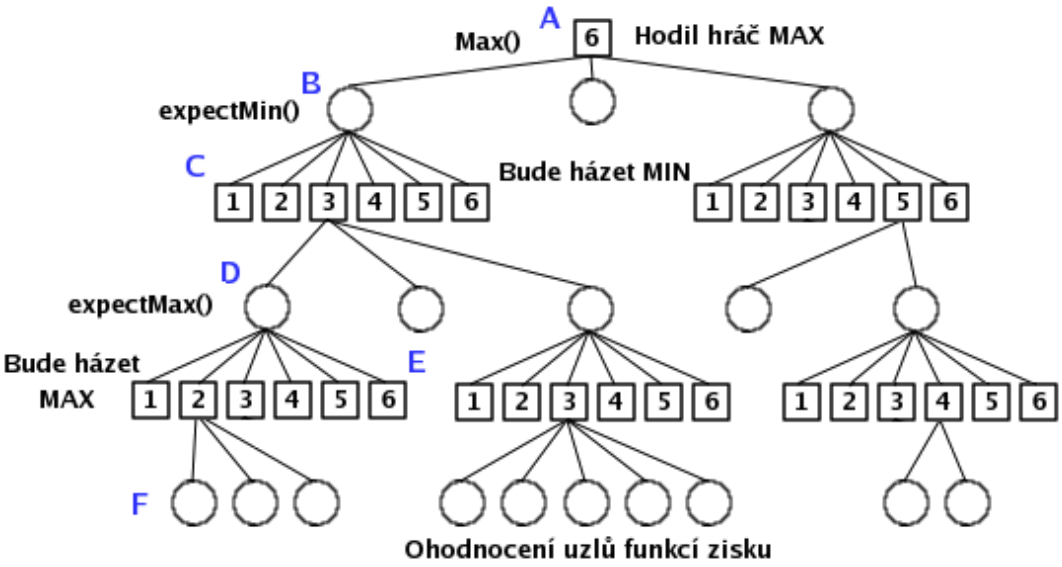
hráče MIN (uzel B), který vybírá minimum, je již teď vidět, že si uzel D nevybere, protože hodnota, která se z uzlu D vrátí nebude nižší než 9. Proto se prohledávání v uzlu D ukončí a vrátí se hodnota 9, kterou ale hráč MIN (uzel B) vybírající minimum neakceptuje.

Stejný princip se uplatní v uzlu F, kde si hráč MIN vybírá minimum a kde již po prohledání prvního podstromu (uzel G) je pro hráče MAX (uzel A) zřejmé, že větev začínající uzlem F je méně atraktivní než větev začínající uzlem B, čímž dojde k "odříznutí" podstromů majících kořeny v uzlech H a I.

3.3.4 Hry s neurčitostí

Dalším typem her, v nichž opět hrají dva protihráči, kteří se po jednotlivých tazích hry pravidelně střídají, mají úplnou informaci o stavu hry, hrají čestně a oba si přejí zvítězit, jsou hry s neurčitostí. Jediným a podstatným rozdílem je, že se při hraní těchto her používá kostka, respektive kostky, čímž do hry vstupuje náhoda - neurčitost.

Základní princip algoritmu ExpectMiniMax je na obrázku 3.6. Hráč MAX hodil kostkou číslo šest, hodnota na kostce však není pro další postup podstatná.



Obrázek 3.6: Základní princip algoritmu ExpectMiniMax

Hráč MAX nyní ví, které své tahy na úrovni B může uskutečnit. A z těchto tahů si bude vybírat uzel s maximální (funkce Max()). Vyjde z úvahy, že hráč MIN by pro známý výsledek hodu vybral tah do stavu (úroveň D) s minimálním ohodnocením. Hráč MIN však výsledek svého hodu nezná, a proto může pracovat pouze s ohodnocením očekávaným, které je na obrázku označeno jako *expectMin()* (očekávané minimum):

$$expectMin() = \sum P(h) * min(D)$$

Kde h je výsledek hodu kostkou (1, 2, 3, 4, 5, 6), $P(h)$ je pravděpodobnost s jakou může na kostce padnout konkrétní hodnota h a $min(D)$ je minimální hodnota z uzlů na úrovni D.

Ohodnocení *expectMin* je tedy dáno součtem ohodnocení po všech možných výsledcích hodu kostky (úroveň C), kdy každé jednotlivé ohodnocení je dáno součinem pravděpodobnosti daného výsledku hodu kostky a následného minimálního ohodnocení stavu, kterého je možné po daném hodu dosáhnout (úroveň D).

Podobným způsobem se postupuje při vyšetřování očekávaného ohodnocení na úrovni D. Hráč MAX vybírá maximum z možných ohodnocení, na obrázku toto hodnocení označeno jako *expectMax()* (očekávané maximum):

$$expectMax() = \sum P(h) * max(F)$$

Kde h je opět výsledek hodu kostkou, $P(h)$ je pravděpodobnost s jakou může na kostce padnout konkrétní hodnota h a $max(F)$ je maximální hodnota z uzlů na úrovni F.

I přesto, že se může zdát algoritmus ExpectMiniMax složitý, je opět snadno realizovatelný rekurzivní procedurou.

Procedura ExpectMiniMax

1. Nazvěme předaný vstupní uzel uzlem X.
2. Je-li uzel X listem (konečným stavem hry, nebo uzlem v maximální hloubce) vrať ohodnocení tohoto uzlu. Jinak pokračuj.
3. Je-li na tahu hráč MAX, tak postupně pro všechny jeho možné tahy (bezprostřední následníky uzlu X a hráče MIN) volej proceduru ExpectMiniMax a vrať maximální hodnotu z hodnot expectMax. Je-li X kořenovým uzlem vrať i tah, který vede k nejlépe ohodnocenému bezprostřednímu následníkovi.
4. Je-li na tahu hráč MIN, tak postupně pro všechny jeho možné tahy (bezprostřední následníky uzlu X a hráče MAX) volej proceduru ExpectMiniMax a vrať minimální hodnotu z hodnot expectMin.

I na algoritmus ExpectMiniMax se dá stejným efektem aplikovat procedura *Alfa-beta prořezávání* (viz 3.5).

Kapitola 4

Praktická část

Tato část dokumentace začíná výběrem hry pro demonstraci metod, které se používají k řešení problematiky neurčitosti. Následuje analýza a návrh řešení. Hlavní kapitoly praktické části tvoří návrh a zpracovnání GUI a popis implementace herního jádra.

4.1 Výběr hry pro demonstraci - Vrhcáby

Vhodné hry pro demonstraci metod řešících problém neurčitostí jsem hledal mezi hrami aleatorickými[1], které jsou založené na principu náhody nebo štěstí nezávislém na vůli jedince.

Mezi aleatorické hry patří *hry v kostky, rulety, loterie, Vrhcáby, člověče nezlob se, domina, Scrablle* apod.

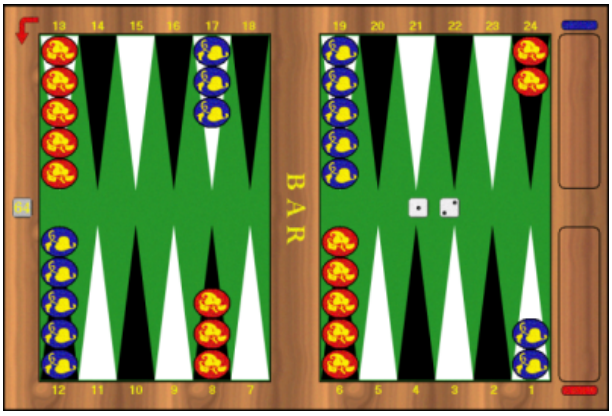
Zaujaly mě především hry *člověče nezlob se* a *Vrhcáby*[2](viz 4.1), v nichž proti sobě hrají dva hráči a snaží se porazit jeden druhého. U těchto her mají oba hráči po celou dobu hry úplnou informaci o jejím stavu a při hře se používá hrací kostka, respektive kostky. Čímž jsou splněny podmínky z úvodu kapitoly 3.3.4 **Hry s neurčitostí**.

Při konečném rozhodování, kterou hru si vybrat, jsem si nakonec zvolil hru *Vrhcáby* z těchto důvodů:

- *Vrhcáby* jsem doposud neznal a pravidla mě velice zaujala, protože se nepodobala žádným, se kterými jsem se doposud setkal
- ač v této hře hraje náhoda podstatnou roli, na rozdíl od *člověče nezlob se* výsledek hry mnohem více závisí na zkušenosti hráče a na použité strategii

[1] *Aleatorické hry* - z latinského *alea*, což v překladu znamená kostka

[2] *Vrhcáby* - anglicky Backgammon - je jedna z nejstarších známých deskových her, předpokládá se, že se hrála již ve starověkém Egyptě, Sumeru, Mezopotámii či Persii.



Obrázek 4.1: Hra Vrhcáby (Backgammon)

4.2 Analýza problému

Hlavní problém projektu naimplementovat hru Vrhcáby v jazyce C/C++ s grafickým uživatelským rozhraním jsem si rozdělil na dva základní podproblémy, podle nichž je koncipována i tato dokumentace. A sice na implementaci GUI a herního jádra. Stěžejním úkolem je naprogramovat herní jádro, respektive UI počítače, nic méně celková grafická podoba programu pro mě má též velký význam.

Také jsem se rozhodl zahrnout možnost přeložit program jak na platformě Linux, tak i v operačním systému Windows XP.

Základní verze hry by měla umožňovat hru hráče proti počítači. Celkový problém implementace je zapotřebí dále rozdělit na dílčí podproblémy jako např. uživatelské ovládání programu či jeho interakce. Je též potřeba vyřešit jednotlivé fáze hry, jako je nalezení korektních tahů pro hráče na tahu, realizace těchto tahů atd. S programováním UI počítače, souvisí i výběr některé z metod pro řešení úloh s neurčitostí. Také k tvorbě GUI je nutné si vybrat vhodný nástroj.

S tvorbou GUI souvisí i problém stanovit jakým způsobem budou reprezentovány jednotlivé objekty na hrací ploše a jak bude řešena jejich správa (vykreslováni, pohyb objektu apod.).

4.2.1 Vlastní řešení

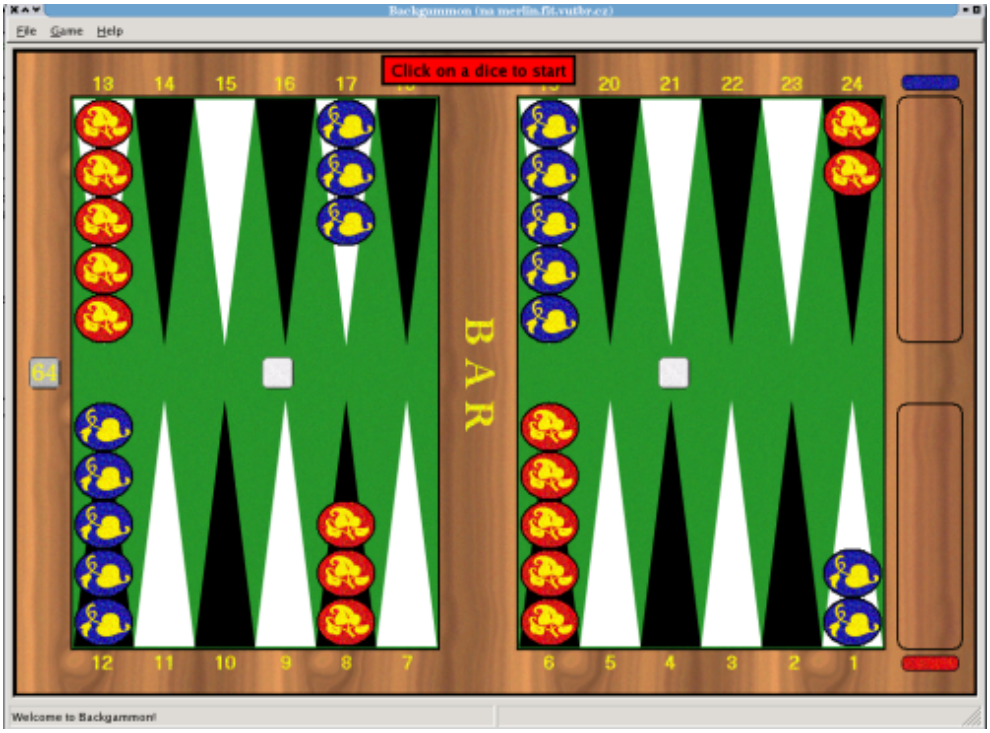
Pro tvorbu programu jsem si zvolil implementační prostředí Linux. V OS Windows XP jsem si nakonfiguroval prostředí MSYS s použitím překladače MINGW, kde jsem si překládal zdrojové soubory vytvořené v Linuxu a následně kontroloval kompatibilitu programu s OS Windows.

Pro vytvoření GUI jsem si na základě dřívější zkušenosti vybral toolkit WxWidgets. V rozhodování mi pomohla i skutečnost, že toolkit WxWidgets je multiplatformní.

Vlastní program tvoří dva hlavní objekty. A sice hlavní okno, objekt *frame*, obsahující herní menu, stavový řádek a hrací plochu. Hrací plocha je zastoupena objektem *scene*. Hra se ovládá výhradně pomocí myši, nepočítám-li klávesové zkratky pro rychlejší přístup k položkám menu.

4.3 Grafické uživatelské rozhraní

Jak jsem uvedl výše, pro implementaci GUI jsem si vybral toolkit WxWidgets, ve kterém jsem implementoval základní objekt programu *frame*. Tento objekt po spuštění aplikace vytvoří hlavní okno s herním menu, stavovým řádkem a vytvoří i objekt hrací plochy, kde probíhá vlastní hra. Na obrázku 4.2 je vidět úvodní okno zobrazené po startu aplikace.



Obrázek 4.2: Úvodní obrazovka zobrazená po spuštění hry

Vykreslení hrací plochy probíhá ve funkci *OnPaint()*, která je volána při zneplatnění hrací plochy, většinou některé její části. Zneplatněním se myslí např. přesun hlavního okna nebo jeho překrytí jiným oknem a následné zobrazení. V těchto případech je třeba překreslit ta část okna, která byla narušena (zneplatněna). Zneplatnění se dá účelně vyvolat i v samotném programu. Např. po té co byl proveden tah hráče.

Překreslování okna při změně stavu hry probíhá následujícím způsobem:

Nejprve se zavolá funkce *RedrawBuffer()*, která do pomocného bufferu (bitmapy) nejprve zakreslí prázdnou hrací plochu a poté vykreslí jednotlivé hrací kameny. Na závěr vykreslí zbylé objekty scény (hrací kostky, sázecí kostka, šipka ukazující směr hry).

Následným voláním funkce *Refresh()*, která je členskou metodou třídy wxWindow, z níž je odvozen objekt *scene*, se zneplatní obsah hrací plochy. Na tuto událost aplikace reaguje zavoláním fce *OnPaint()* zmiňované výše. Funkce *OnPaint()* již pouze vykreslí pomocný buffer do device kontextu okna.

Během hry program pomocí zpráv reaguje na špatný vstup uživatele (pokus o nesprávný tah) nebo radí, co má uživatel udělat. Na obrázku 4.2) je vidět zpráva v červeném rámečku.

Vzhled všech objektů ve hře je realizován pomocí obrázku ve formátu png. Všechny obrázky jsou načteny po startu aplikace z adresáře *graphics*.

4.3.1 Reprezentace hrací plochy

Jedním z hlavních problémů, který jsem musel řešit, je samotné uchování stavu hry. To znamená jednotlivé rozložení hracích kamenů.

Původní návrh objektu hrací plochy spočíval v reprezentaci jednotlivých trojúhelníkových ploch (klínů) jako samostatných objektů. Tyto uchovávaly počet a typ hracích kamenů umístěných na klínu. Později při implementaci metod manipulujících s těmito objekty, jako nalezení možných tahů, realizace tahu nebo při generování prohledávacího stromu se tyto objekty ukázaly nepřehledné.

Z tohoto důvodu jsem hrací plochu implementoval jako jednorozměrné homogenní pole, jehož prvky reprezentují výše zmiňované klíny. Tato struktura se ukázala jednak přehlednější, tak i méně náročná na paměť a v neposlední řadě i rychlejší, co se týká přístupu k jednotlivým klínům a manipulace s nimi. Pole je složeno z 28 základních prvků a 6 prvků použitých při generování prohledávacího stromu.

Tuto základní strukturu je možné vidět na obrázku 4.3. Jednotlivé prvky pole mají tento význam:

0 - uchovává počet kamenů na baru pro hráče ovládaného počítačem

1 - 24 - uchovává počet kamenů na očíslovaných klínech ve hře

25 - uchovává počet kamenů na baru pro hráče (uživatelem)

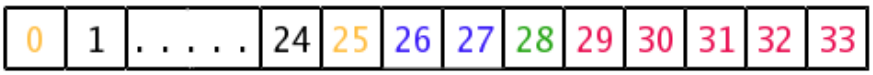
26 - reprezentuje pole, kam hráč vyvádí své kameny

27 - reprezentuje pole, kam počítač vyvádí své kameny

28 - příznak vyhození protihráčova kamene

29 - údaj o první kostce použité pro tah

30 - 32 - údaj ze kterého klínu bylo kostkou 1 - 4 táž

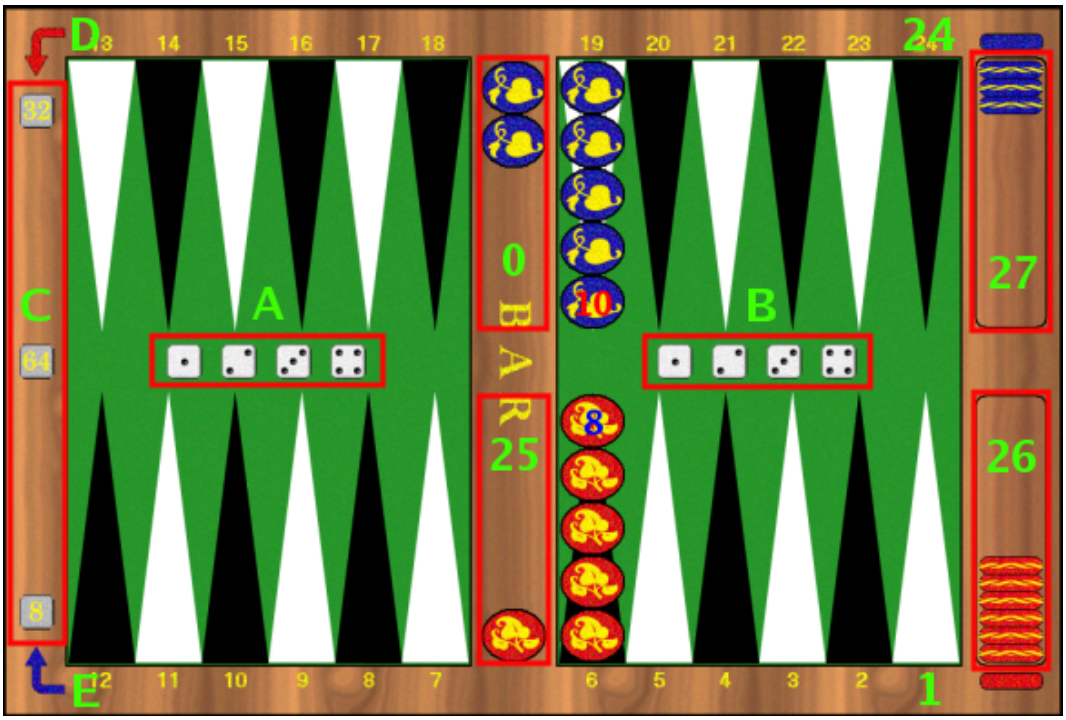


Obrázek 4.3: Struktura uchovávající aktuální stav hry

Základní prvky pole

Na obrázku 4.4 jsou zvýrazněny výše zmiňované základní prvky pole s indexy 0 - 27. Přičemž u prvků 1 - 24 jsou zvýrazněny jen krajní hodnoty. Index jednotlivých klínů se zvyšuje od prvku 1 ve směru hodinových ručiček až k prvku 24. Informace, kterou tyto základní prvky uchovávají, je počet kamenů na daném klínu. Jak je vidět na obrázku na jednom klínu může být zobrazeno maximálně pět hracích kamenů, ale ve hře jich může stát na jednom klínu všech 15 najednou. Toto je při vykreslování vyřešeno číslicí udávající skutečný počet kamenů umístěných na klínu, která se zobrazuje v okamžiku, kdy počet kamenů překročí 5.

I přesto, že je toto pole jednorozměrné, uchovává každý ze základních prvků dvě informace. První je samozřejmě počet kamenů umístěných na daném klínu a druhá je typ kamene, kde typ říká, kterému hráči kameny patří. Toto je implementováno rozdílným znaménkem u počtu kamenů. Počet kamenů hráče je uchováván s kladnou hodnotou a kameny protihráče (hráče ovládaného počítačem) se zápornou hodnotou. Podle typu se určuje i vzhled kamene.



Obrázek 4.4: Hrací plocha s vyznačenými částmi

Speciální prvky pole

Prvním ze speciálních prvků je prvek s indexem 28 reprezentující příznak, který se nastaví v případě, vede-li pohyb hracího kamene hráče na políčko obsazené soupeřovým kamenem, jinými slovy došlo-li k vyhození soupeřova kamene. Tento příznak se používá při ohodnocování terminálního stavu v prohledávacím stromu, kdy stav hry, v němž došlo k vyhození soupeřova kamene, je hodnocen pomocí jiných hodnot, než kdyby došlo k obyčejnému přesunu kamene.

Všechny další speciální prvky spolu úzce souvisí. Jsou totiž všechny použity při rekonstrukci tahu počítače, poté co je nalezen nejlepší z tahů. První s indexem 29 říká, která z kostek byla použita při realizaci tahu jako první. Zbylé čtyři prvky obsahují index klínu, ze kterého bylo pomocí dané kostky taženo s tím, že první kostku zastupuje prvek s indexem 30, druhou prvek s indexem 31 atd. Problematika hracích kostek je detailně popsána v následující podkapitole.

4.3.2 Reprezentace hracích kostek

Další objekty, kterými jsou hrací kostky, sázecí kostka a šipky určující směr hry jsou již samostatné a nesou si informaci o umístění na hrací ploše a informaci o svém vzhledu.

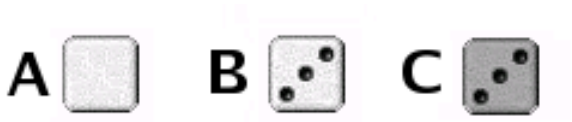
Z obrázku 4.4 je vidět skutečný počet vytvořených objektů a jejich umístění na hrací ploše.

Jedná se o dvě pomocné šipky označené v obrázku písmeny D a E, z nichž maximálně jedna může být zobrazena na hrací ploše. Červená šipka (D) ukazuje směr hry pro hráče uživatele a je zobrazena vždy po čas jeho tahu. Modrá šipka pracuje obdobně pro hráče ovládaného počítačem a má spíše informativní charakter pro hráče uživatele, že zrovna není na tahu.

Dalším objektem je sázecí kostka (C), jejíž funkce bude implementována až pro hraní série her, při nichž se počítá skóre. Tato kostka určuje násobek základní hodnoty výhry (viz 5) a může nabývat hodnot 2, 4, 8, 16, 32 a 64. Zde je tato kostka reprezentována celkem třemi kostkami, z nichž právě jedna je vždy zobrazena. Její poloha je určena tím, který z hráčů si naposledy vsadil na svoji výhru.

Na konec jsem si nechal hrací kostky používané během hry ze všech výše zmíněných objektů nejčastěji. Ve skutečnosti se při hře používají pouze dvě kostky. Pro rozlišení hráče, který je zrovna na tahu, se tyto dvě kostky zobrazují střídavě v levé (A) a pravé části (B) herní plochy. V levé pro hráče ovládaného počítačem a v pravé pro hráče uživatele. Tyto kostky jsou na obrázku 4.4 zobrazeny s hodnotami 2 a 3. Zbylé dvě (čtyři) kostky se zobrazují pouze v případě, kdy padne na obou kostkách stejná hodnota. Při tomto hodu totiž pravidla říkají, že hráč má k dispozici další dva tahy stejné hodnoty navíc.

Hrací kostky se objevují ve třech základních podobách (viz obrázek 4.5). První vzhled, kdy kostky nemají žádnou hodnotu (A) a čekají na hráče, který kliknutím do hrací plochy realizuje jejich vrh. Po "hodu" se zobrazí již se svou aktuální hodnotou (B). Poslední možný vzhled se objeví na kostce po odehrání její hodnoty (C).



Obrázek 4.5: Zobrazení hracích kostek ve hře

4.4 Herní jádro

Objekt *scene* se stará o vykreslování hrací plochy a jednotlivých objektů, jak popisuje předchozí kapitola, ale i o vlastní hraní hry, což bude popsáno v této kapitole.

Při hře se střídá 6 základních stavů (fází) hry. Jsou to:

- počáteční stav hry
- fáze hodu kostkou (na tahu je hráč uživatel)
- fáze tahu hráče (uživatele)
- fáze hodu kostkou (na tahu je počítač)
- fáze tahu hráče (počítače)
- konečný stav hry (jednomu z hráčů se podařilo zvítězit)

4.4.1 Počáteční stav hry

Počáteční stav hry reprezentován fcí *NewGameHandler()* je vidět na obrázku 4.2, kde aplikace čeká na kliknutí uživatele na hrací plochu. Po té, co se tak stane, se pomocí generátoru pseudonáhodných čísel vygeneruje hodnota pro každou z kostek. Podle toho, která z hodnot je vyšší, zda levá či pravá, se přejde buď do fáze hodu kostkou hráče, pro vyšší pravou hodnotu, nebo do fáze hodu kostkou počítače v opačném případě.

Počáteční stav hry tedy pouze vygeneruje hod kostkami a určí, který z hráčů započne hru.

4.4.2 Fáze hodu kostkou

Fáze hodu kostkou reprezentovaná fcí *P1RollHandler()* nebo *P2RollHandler()*, v případě, že ji předcházel počáteční stav, přejímá hodnotu na kostkách vygenerovanou v počátečním stavu nebo sama vygeneruje nové hodnoty. Postará se prostřednictvím funkcí *SetDiceValue()* a *GetDiceLook()* o zobrazení správných hodnot.

Následně zavolá funkci *FindPossPlMoves()* (v případě že táhne hráč) nebo funkci *FindPossPcMoves()*, která vyhledá na základě pravidel a aktuálním stavu hry všechny přípustné tahy pro hodnotu první hrací kostky. Poté se změní stav hry na fázi tah hráče a čeká se na hráčův vstup, v případě, že je na tahu hráč. Nebo se po krátkém intervalu samovolně přejde do fáze tahu hráče počítače, aby to stihl hráč postřehnout. Neexistuje-li žádný korektní tah pro první hodnotu, opakuje se zmíněný postup pro hodnotu na druhé kostce a v případě nalezení alespoň jednoho tahu se přejde do stavu fáze tahu příslušného hráče s informací, že první tah bude realizován pomocí druhé hodnoty. Pokud se ani pro druhou hodnotu nenajde přípustný tah, změní se stav hry na fázi hodu kostkou protihráče.

4.4.3 Fáze tahu hráče

V této fázi se obsluha pro hráče a počítač natolik liší, že je popíší v samostatných podkapitolách.

Fáze tahu hráče uživatele

Obsluhu této fáze zajišťuje funkce *P1MoveHandler()*, které se předá parametr obsahující souřadnice bodu, do kterého hráč kliknul levým tlačítkem myši. Toto je důležité pro další fázi obsluhy. Kliknul-li hráč na jinam než na klín, ze kterého je možné táhnout, je na to upozorněn zprávou v horní části okna, která po dvou vteřinách zmizí. Kliknul-li na klín, ze kterého je možný tah, tento se provede a dojde k překreslení scény a vyhodnotí se další možný tah pro druhou kostku. Pokud takový tah neexistuje, přejde se do fáze hodu kostkou protihráče.

Fáze tahu hráče počítače

Obsluhu této fáze zajišťuje funkce *P2MoveHandler()*. V této fázi je známa hodnota kostky, pro kterou existuje alespoň jeden přípustný tah. A proto první věc, která se provede, je vygenerování stavů hry po všech možných tazích, o co se stará funkce *CreateNodes()*. Tyto stavy jsou uloženy do vektoru a v vzápětí budou tvořit kořenové uzly vygenerovaných prohledávacích stromů, při hledání nejlépe ohodnoceného tahu.

Dále následuje již samotné volání funkce *ExpectMiniMax()* (viz 3.3.4) postupně pro všechny vygenerované uzly. Funkce *ExpectMiniMax()* po průchodu prohledávacím stromem do definované hloubky vrací ohodnocení předaného uzlu, z něhož vybere maximum. V poli, ve kterém je uložen stav hry po nejlépe ohodnoceném tahu je uložena i informace nutná pro rekonstrukci tahu (viz 4.3).

Následujícím krokem, jak jsem již předeslal, je rekonstrukce nejlépe hodnoceného tahu. Realizují se tahy pro jednotlivé hodnoty na hracích kostkách a to s nutnou prodlevou, aby bylo hráči uživateli zřejmé, jak počítač táhnul.

Realizace algoritmu ExpectMiniMax()

Algoritmus ExpectMiniMax je realizován podle pokynů uvedených v teoretické části této dokumentace (viz 3.3.4). Funkce zisku zastoupená ve zdrojovém kódu funkcí *Eval()* je implementována dle vzoru uvedeného na webových stránkách (viz [5]). Je založena na ohodnocení stavu hry po ukončení tahu, s tím, že ohodnocení jednoho uzlu (stavu hry) spočívá v součtu všech ohodnocení jednotlivých klínů, na kterých se nachází nějaký kámen. Ohodnocení je závislé na počtu kamenů se na daném klínu a také na tom zda při tahu došlo k "vyhození" protihráčova kamene. K ohodnocení dílčích klínu jsou použity konstanty uvedené v souboru *weight.h*.

Kapitola 5

Závěr

Po nastudování a především pochopení teorie řešení úloh s neurčitostí pomocí algoritmu ExpectMiniMax jsem si jako demonstrační řešení vybral implementaci hry Vrhcáby.

Hru se mi podařilo naimplementovat do stavu, ve kterém je plně hratelná. Bohužel z nedostatku času, který jsem musel rozdělit mezi více projektů řešených prakticky paralelně, jsem nestihl naprogramovat některá rozšíření, či vylepšení, která mě napadala během času, který jsem tvorbou této práce strávil. Rozšíření se týkají prakticky všech základních částí aplikace. Např. v části GUI by možným rozšířením mohla být volba vzhledu hrací desky, či hracích kamenů. Nebo zvýraznění kamene, kterým se právě táhne. V části herního jádra by to mohlo být rozšíření týkající se hodnotící funkce, či možnost zvyšování hodnoty hry pomocí sázecí kostky, s čímž je spojena možnost hrát sérii her a uchovávat výsledky pro různé hráče. Zajímavým vylepšením by mohla být i implementace možnosti kroku zpět či ukládání rozehrané partie.

I přesto, že spousta možností jak rozšířit hru zůstala jen na papíře, mi tento projekt mnohé dal. Pro vytvořeni vzhledu jednotlivých objektů jsem se např. naučil základy Photoshopu. Získal jsem nové zkušenosti s použitím typografického nástroje LaTeX, ve kterém je vysázena tato dokumentace. Samotné řešení problému implementovat hru s umělou inteligencí, navíc s prvkem náhody, mě v mnohém obohatilo. Nemalý význam také dávám získaní nových zkušeností při plánování času pro souběžné řešení více projektů.

Hra Vrhcáby je po hře Piškvorky, kterou jsem implementoval ještě jako student střední školy, teprve druhou hrou, kterou se mi podařilo naprogramovat. Proto bych rád v budoucnu navázal na získané zkušenosti v oblasti umělé inteligence a pokusil se naimplementovat nějakou z dalších zajímavých her.

Literatura

[1] F. V. Zbořil a F. Zbořil. Základy umělé inteligence [studijní opora], v.3 2006 [citováno 2007]. Publikace přístupná pouze osobám přihlášeným do predmětu IZU, přístupné z https://www.fit.vutbr.cz/study/courses/IZU/.

[2] V. Mařík a kol. *Umělá inteligence 1.* Academia, Praha, 1993. ISBN:80-200-0496-3.

[3] Z. Kotek a kol. *Metody rozpoznávání a umělá inteligence.* ČSVTS FE VŠSE, Plzeň, 1983. Citováno dle [2].

[4] M. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967. Citováno dle [2].

[5] WWW stránky. Benchmark player 'pubeval.c'. `http://www.bkgm.com/archive.html. kategorie 'Source Code'.`

`[6] WWW stránky. Vrhcáby. http://cs.wikipedia.org/wiki/Vrhc%C3%A1by.`

Příloha A

Pravidla hry Vrhcáby [6]

Tahy

Na začátku hry hodí každý hráč jednou kostkou a ten hráč, který hodil větší číslo (při stejných hodnotách se hází znovu), začíná hru tím, že odehraje obě hozená čísla. V dalších kolech již hází každý hráč oběma kostkami sám. Hod se bere jako dvě oddělená čísla, určující počet polí, o které smí hráč postoupit zvolenými kameny. Hráč může postoupit jedním kamenem o součet obou čísel, ale pouze tak, že nejprve posune kámen o hodnotu jedné kostky (což musí být sám o sobě platný tah, tzn. kámen nesmí skončit na poli obsazeném více než jedním soupeřovým kamenem), teprve poté o hodnotu druhé kostky. Pokud na obou kostkách padne stejné číslo, bere se dvojnásobně, tzn. jako by padlo na čtyřech kostkách. Tyto čtyři hodnoty hráč může rozdělit na posun čtyř různých kamenů, na posun jednoho kamene o čtyřnásobek hozené hodnoty (ovšem opět ve čtyřech nezávislých krocích!), nebo libovolnou kombinaci těchto možností.

Hráč se nesmí vzdát svého tahu, třebaže je pro něj nevýhodný. Pokud existuje nějaký tah splňující pravidla, musí hráč táhnout. Pokud hráč může legálně hrát pouze část tahu (např. pouze hodnotu jedné kostky), musí odehrát co největší část tahu splňující pravidla. Pokud hráč může odehrát libovolnou hodnotu z obou kostek, ale nemůže hrát obě, musí hrát tu vyšší.

Vyhazování a vracení do hry

Jak už bylo zmíněno, pokud kámen skončí tah (nebo jeho část danou posunem o hodnotu na jedné kostce) na poli, obsazeném jedním soupeřovým kamenem, tento kámen vyhodí, tzn. přemístí na bar (také zvaný přepážka či závora), což je vyvýšené místo uprostřed desky. Bar se chápe jako pole umístěné před začátkem desky, nejvzdálenější od cíle. Když má hráč nějaký kámen (nebo kameny, počet kamenů na baru není omezen) na baru a chce je dostat zpět do hry, provede to přesně tak, jako by daný kámen opravdu stál jedno pole před deskou, tzn. např. hodem 1 přesune kámen na první pole desky (pochopitelně pokud není obsazeno soupeřem). Dokud má hráč alespoň jeden kámen na baru, musí tyto kameny dostat do hry před tím, než hraje jiným kamenem. Pokud mu hod kostky toto neumožňuje, hod mu propadá a hráč nehraje (popř. hraje pouze hodnotu na druhé kostce).

Pokud je na některém poli dva nebo více kamenů, je pole obsazeno a soupeř na tomto poli nesmí ukončit tah ani jeho část. Z tohoto důvodu je zřejmé, že šest obsazených polí v řadě je pro soupeře nepřekročitelnou překážkou (této formaci se říká prima). Pokud se navíc podaří hráči obsadit všech šest polí ve své domácí ohradě (tzn. v tom kvadrantu desky, ze kterého vyvádí kameny, což je současně kvadrant, do kterého soupeř nasazuje kameny vracející se z baru) v době, kdy má soupeř nějaký kámen na baru, soupeřovi se nemůže

podařit vrátit tento kámen do hry, takže nemůže hrát a dokud tato situace trvá (dokud není prima rozpuštěna), ani nehází kostkami.

Vyvádění kamenů

Ve chvíli, kdy jsou všechny kameny hráče umístěny v jeho domácí ohradě (na posledních šesti polích desky), může hráč vyvádět kameny mimo desku. To provádí tak, že kameny jakoby táhne na fiktivní pole těsně za deskou, tzn. pokud na kostce padlo např. číslo čtyři, odstraní hráč jeden kámen ze čtvrtého pole. Pokud hráči padne číslo vyšší, než je jeho nejvzdálenější kámen, odstraní jeden z kamenů na nejvzdálenějším poli. (Tato poučka se netýká situace, kdy na daném poli sice neleží žádný kámen, ale nějaký kámen je ještě za daným polem. Tehdy musí hráč provést běžný tah kamenem po desce.) Pokud se v průběhu této koncovky stane, že se některý z kamenů dostane mimo domácí ohradu (soupeř ho vyhozením pošle na bar), hráč nemůže pokračovat ve vyvádění kamenů do té doby, než jsou opět všechny jeho zbylé kameny v domácí ohradě.

Hra končí ve chvíli, kdy jeden z hráčů vyvede z desky poslední kámen, čímž se stává vítězem. Hru je samozřejmě také možné ukončit v jejím průběhu (před hodem hráče) tím, že hráč nabídne soupeři svou rezignaci, kterou soupeř může a nemusí přijmout.

Příloha B

Adresářová struktura na přiloženém CD

- bakalarskaPrace
  - doc - dokumentace bakalářské práce
    - cls - sablona fitthesis.cls a obrázky v ní použité
    - fig - obrázky pro dokumentaci
  - graphics - obrázkeky všech objektu na hrací ploše
  - include - hlavičkové soubory
  - src - zdrojové soubory

Postup při překldau programu

Program je přeložitelný jak v OS Linux, tak v OS Windows XP, např. za použití systému MSYS v kombinaci s překladačem MinGW. Jediný požadavek na systém je nainstalovaný GUI toolkit WxWidgets GTK verze 2.8.0 nebo výšší. Překlad se spustí z příkazové řádky příkazem make. Vznikne spustitelný soubor backgammon (Linux) nebo backgammon.exe (Windows XP).

Stručný návod k použití

Před spuštěním hry, doporučuji prostudovat pravidla hry Vrhcáby, která jsou obsahem přílohy A.

Po spuštění aplikace začněte hru kliknutím levým tlačítkem myši do hrací plochy. Při hře jsou uživateli vypisovány základní zprváy, které mu radí co v dané situaci dělat. Hru je možné myší, ovládání je zcela intuitivní. Pro hod kostkou klikněte na kostky, poku jste na tahu objeví se na nich hodnota se kterou můžete táhnout. Pro přesun kamene klikněte na kámen, kterým chcete táhnout. Korektnost tahu je automaticky kontrolována. Pro přesun se použije vždy první kostka s hodnotou, pro kterou existuje korektní tah.